Neural Networks
Reminders

• Post-Exam Followup:
  – Exam Viewing
  – Exit Poll: Exam 1
  – Grade Summary 1

• Homework 4: Logistic Regression
  – Out: Fri, Feb 18
  – Due: Sun, Feb. 27 at 11:59pm

• Swapped lecture/recitation:
  – Lecture 12: Fri, Feb. 25
OPTIMIZATION FOR L1 REGULARIZATION
Optimization for L1 Regularization

Can we apply SGD to the LASSO learning problem?

\[ \arg\min_{\theta} J_{\text{LASSO}}(\theta) \]

\[ J_{\text{LASSO}}(\theta) = J(\theta) + \lambda \|\theta\|_1 \]

\[ = \frac{1}{2} \sum_{i=1}^{N} (\theta^T x^{(i)} - y^{(i)})^2 + \lambda \sum_{k=1}^{K} |\theta_k| \]
Optimization for L1 Regularization

• Consider the absolute value function:

\[ r(\theta) = \lambda \sum_{k=1}^{K} |\theta_k| \]

• The L1 penalty is subdifferentiable (i.e. not differentiable at 0)

**Def:** A vector \( g \in \mathbb{R}^M \) is called a **subgradient** of a function \( f(x) : \mathbb{R}^M \rightarrow \mathbb{R} \) at the point \( x \) if, for all \( x' \in \mathbb{R}^M \), we have:

\[ f(x') \geq f(x) + g^T (x' - x) \]
Optimization for L1 Regularization

• The L1 penalty is subdifferentiable (i.e. not differentiable at 0)
• An array of optimization algorithms exist to handle this issue:
  – Subgradient descent
  – Stochastic subgradient descent
  – Coordinate Descent
  – Othant-Wise Limited memory Quasi-Newton (OWL-QN) (Andrew & Gao, 2007) and provably convergent variants
  – Block coordinate Descent (Tseng & Yun, 2009)
  – Sparse Reconstruction by Separable Approximation (SpaRSA) (Wright et al., 2009)
  – Fast Iterative Shrinkage Thresholding Algorithm (FISTA) (Beck & Teboulle, 2009)

Basically the same as GD and SGD, but you use one of the subgradients when necessary
NEURAL NETWORKS
1. Given training data:
   \[ \{ x_i, y_i \}_{i=1}^{N} \]

2. Choose each of these:
   - Decision function
     \[ \hat{y} = f_{\theta}(x_i) \]
   - Loss function
     \[ \ell(\hat{y}, y_i) \in \mathbb{R} \]

**Examples:** Linear regression, Logistic regression, Neural Network

**Examples:** Mean-squared error, Cross Entropy
1. **Given training data:**
\[ \{ \mathbf{x}_i, y_i \}_{i=1}^{N} \]

2. **Choose each of these:**
   - Decision function
     \[ \hat{y} = f_\theta(\mathbf{x}_i) \]
   - Loss function
     \[ \ell(\hat{y}, y_i) \in \mathbb{R} \]

3. **Define goal:**
\[ \theta^* = \arg \min_{\theta} \sum_{i=1}^{N} \ell(f_\theta(\mathbf{x}_i), y_i) \]

4. **Train with SGD:**
(take small steps opposite the gradient)
\[ \theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_\theta(\mathbf{x}_i), y_i) \]
A Recipe for Machine Learning

1. Given training data:
\[ \{x_i, y_i\}_{i=1}^{N} \]

2. Choose each of these:
   - Decision function
   \[ \hat{y} = f_{\theta}(x_i) \]
   - Loss function
   \[ \ell(\hat{y}, y_i) \in \mathbb{R} \]

3. Define goal:

4. Train with SGD:
\[ \theta(t+1) = \theta(t) - \eta_t \nabla \ell(f_{\theta}(x_i), y_i) \]

**Gradients**

Backpropagation can compute this gradient!

And it’s a **special case of a more general algorithm** called reverse-mode automatic differentiation that can compute the gradient of any differentiable function efficiently!
A Recipe for Machine Learning

1. Given training data:
2. Choose each of these:
   – Decision function
   – Loss function
3. Define goal:
4. Train with SGD:
   (take small steps opposite the gradient)

Goals for Today’s Lecture

1. Explore a **new class of decision functions** (Neural Networks)
2. Consider **variants of this recipe** for training

\[
\hat{y} = f_\theta(x_i)
\]

\[
\ell(\hat{y}, y_i) \in \mathbb{R}
\]

\[
\theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_\theta(x_i), y_i)
\]
Decision Functions

Linear Regression

\[ y = h_\theta(x) = \sigma(\theta^T x) \]

where \( \sigma(a) = a \)
Logistic Regression

$$y = h_{\theta}(x) = \sigma(\theta^T x)$$

where $\sigma(a) = \frac{1}{1 + \exp(-a)}$
Decision Functions

Perceptron

\[ y = h_\theta(x) = \sigma(\theta^T x) \]
where \( \sigma(a) = \text{sign}(a) \)
Neural Network

Decision Functions

Output

Hidden Layer

Input

$y$

$X_1$

$X_2$

$X_3$

$X_M$

$Z_1$

$Z_2$

$Z_D$

$\ldots$

$\ldots$
COMPONENTS OF A NEURAL NETWORK
Suppose we already learned the weights of the neural network.

To make a new prediction, we take in some new features (aka. the input layer) and perform the feed-forward computation.
The computation of each neural network unit resembles binary logistic regression.
The computation of each neural network unit resembles binary logistic regression.

\[ .80 = \sigma(1.4) \]

\[ 1.4 = 13(-.4) + 2(.5) + 7(.8) \]
The computation of each neural network unit resembles binary logistic regression.
The computation of each neural network unit resembles binary logistic regression.
The computation of each neural network unit resembles binary logistic regression. Except we only have the target value for y at training time! We have to learn to create “useful” values of $z_1$ and $z_2$ in the hidden layer.
From Biological to Artificial

The motivation for Artificial Neural Networks comes from biology...

Biological “Model”

- **Neuron:** an excitable cell
- **Synapse:** connection between neurons
- A neuron sends an **electrochemical pulse** along its synapses when a sufficient voltage change occurs
- **Biological Neural Network:** collection of neurons along some pathway through the brain

Biological “Computation”

- Neuron switching time: \( \sim 0.001 \text{ sec} \)
- Number of neurons: \( \sim 10^{10} \)
- Connections per neuron: \( \sim 10^{4-5} \)
- Scene recognition time: \( \sim 0.1 \text{ sec} \)

Artificial Model

- **Neuron:** node in a directed acyclic graph (DAG)
- **Weight:** multiplier on each edge
- **Activation Function:** nonlinear thresholding function, which allows a neuron to “fire” when the input value is sufficiently high
- **Artificial Neural Network:** collection of neurons into a DAG, which define some differentiable function

Artificial Computation

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed processes

Slide adapted from Eric Xing
DEFINING A 1-HIDDEN LAYER NEURAL NETWORK
Neural Networks

Chalkboard

– Example: Neural Network w/1 Hidden Layer
**Decision Functions**

**Neural Network**

\[ y = \sigma(\beta_1 z_1 + \beta_2 z_2) \]

\[ z_2 = \sigma(\alpha_{21} x_1 + \alpha_{22} x_2 + \alpha_{23} x_3) \]

\[ z_1 = \sigma(\alpha_{11} x_1 + \alpha_{12} x_2 + \alpha_{13} x_3) \]
Neural Network

\[ y = \sigma(\beta_1 z_1 + \beta_2 z_2) \]

\[ z_2 = \sigma(\alpha_{21} x_1 + \alpha_{22} x_2 + \alpha_{23} x_3) \]

\[ z_1 = \sigma(\alpha_{11} x_1 + \alpha_{12} x_2 + \alpha_{13} x_3) \]
Decision Functions

Neural Network

Output

Weights

Hidden Layer

Weights

Input

\[ y = \sigma(\beta_1 z_1 + \beta_2 z_2) \]

\[ z_2 = \sigma(\alpha_{21} x_1 + \alpha_{22} x_2 + \alpha_{23} x_3) \]

\[ z_1 = \sigma(\alpha_{11} x_1 + \alpha_{12} x_2 + \alpha_{13} x_3) \]
Neural Network

y = \sigma(\beta_1 z_1 + \beta_2 z_2)

z_2 = \sigma(\alpha_{21} x_1 + \alpha_{22} x_2 + \alpha_{23} x_3)

z_1 = \sigma(\alpha_{11} x_1 + \alpha_{12} x_2 + \alpha_{13} x_3)
Neural Network

\[ y = \sigma(\beta_1 z_1 + \beta_2 z_2) \]

\[ z_2 = \sigma(\alpha_{21} x_1 + \alpha_{22} x_2 + \alpha_{23} x_3) \]

\[ z_1 = \sigma(\alpha_{11} x_1 + \alpha_{12} x_2 + \alpha_{13} x_3) \]
Neural Network

Output

Weights

Hidden Layer

Weights

Input

\[ y = \sigma(\beta^T z) \]
\[ z_2 = \sigma(\alpha_{2,T}^T x) \]
\[ z_1 = \sigma(\alpha_{1,T}^T x) \]
NONLINEAR DECISION BOUNDARIES AND NEURAL NETWORKS
$y = h_\theta(x) = \sigma(\theta^T x)$

where $\sigma(a) = \frac{1}{1 + \exp(-a)}$
\[ y = h_\theta(x) = \sigma(\theta^T x) \]

where \( \sigma \) is the logistic function.

In-Class Example

\begin{align*}
\theta_1 & \quad 1 \\
\theta_2 & \quad 1 \\
\theta_3 & \quad 0 \\
\end{align*}
Neural Networks

Chalkboard

– 1D Example from linear regression to logistic regression
– 1D Example from logistic regression to a neural network
Logistic Regression

\[ y = h_\theta(x) = \sigma(\theta^T x) \]

where \( \sigma(a) = \frac{1}{1 + \exp(-a)} \)

Decision Functions

Input:
- \( x_1 \)
- \( x_2 \)
- \( x_3 \)

Output:
- \( \theta_1 \)
- \( \theta_2 \)
- \( \theta_3 \)

Examples:
- Face
- Face
- Not a face
\[ y = h_\theta(x) = \sigma(\theta^T x) \]

Where
\[ (a) = 1 + 2tT(a) \]
Question:
Suppose you are training a one-hidden layer neural network with sigmoid activations for binary classification.

True or False: There is a unique set of parameters that maximize the likelihood of the dataset above.

Answer:
ARCHITECTURES
Neural Network for Classification

(A) Input
Given $x_i$, $\forall i$

(B) Hidden (linear)
\[ a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \forall j \]

(C) Hidden (sigmoid)
\[ z_j = \frac{1}{1+\exp(-a_j)}, \forall j \]

(D) Output (linear)
\[ b = \sum_{j=0}^{D} \beta_j z_j \]

(E) Output (sigmoid)
\[ y = \frac{1}{1+\exp(-b)} \]
Neural Networks

Chalkboard

– Example: Neural Network w/2 Hidden Layers
– Example: Feed Forward Neural Network (matrix form)
Neural Network Architectures

Even for a basic Neural Network, there are many design decisions to make:

1. # of hidden layers (depth)
2. # of units per hidden layer (width)
3. Type of activation function (nonlinearity)
4. Form of objective function
5. How to initialize the parameters
BUILDING WIDER NETWORKS
Q: How many hidden units, $D$, should we use?

The hidden units could learn to be...

- a selection of the most useful features
- nonlinear combinations of the features
- a lower dimensional projection of the features
- a higher dimensional projection of the features
- a copy of the input features
- a mix of the above
Q: How many hidden units, $D$, should we use?

The hidden units could learn to be...
- a selection of the most useful features
- nonlinear combinations of the features
- a lower dimensional projection of the features
- a higher dimensional projection of the features
- a copy of the input features
- a mix of the above

$D < M$
Q: How many hidden units, $D$, should we use?

The hidden units could learn to be...
- a selection of the most useful features
- nonlinear combinations of the features
- a lower dimensional projection of the features
- a higher dimensional projection of the features
- a copy of the input features
- a mix of the above

$D > M$
In the following examples, we have two input features, \( M=2 \), and we vary the number of hidden units, \( D \).

The hidden units could learn to be:

- A selection of the most useful features
- Nonlinear combinations of the features
- A lower dimensional projection of the features
- A higher dimensional projection of the features
- A copy of the input features
- A mix of the above

\[ D \geq M \]
Examples 1 and 2

DECISION BOUNDARY EXAMPLES
Example #1: Diagonal Band
Example #1: Diagonal Band
Example #1: Diagonal Band

Tuned Neural Network (hidden=2, activation=logistic)
Example #1: Diagonal Band

LR1 for Tuned Neural Network (hidden=2, activation=logistic)
Example #1: Diagonal Band

LR2 for Tuned Neural Network (hidden=2, activation=logistic)
Example #1: Diagonal Band

Tuned Neural Network (hidden=2, activation=logistic)
Example #1: Diagonal Band
Example #2: One Pocket
Example #2: One Pocket
Example #2: One Pocket

Tuned Neural Network (hidden=3, activation=logistic)
Example #2: One Pocket

LR1 for Tuned Neural Network (hidden=3, activation=logistic)
Example #2: One Pocket

LR2 for Tuned Neural Network (hidden=3, activation=logistic)
Example #2: One Pocket

LR3 for Tuned Neural Network (hidden=3, activation=logistic)
Example #2: One Pocket

Tuned Neural Network (hidden=3, activation=logistic)
Example #2: One Pocket

LR1 for Tuned Neural Network (hidden=3, activation=logistic)

LR2 for Tuned Neural Network (hidden=3, activation=logistic)

LR3 for Tuned Neural Network (hidden=3, activation=logistic)

Tuned Neural Network (hidden=3, activation=logistic)
Examples 3 and 4

DECISION BOUNDARY EXAMPLES
Example #1: Diagonal Band

Example #2: One Pocket

Example #3: Four Gaussians

Example #4: Two Pockets
Example #3: Four Gaussians
Example #3: Four Gaussians
Example #3: Four Gaussians

K-NN (k=5, metric=euclidean)
Example #3: Four Gaussians

Tuned Neural Network (hidden=2, activation=logistic)
Example #3: Four Gaussians

LR1 for Tuned Neural Network (hidden=2, activation=logistic)
Example #3: Four Gaussians

LR2 for Tuned Neural Network (hidden=2, activation=logistic)
Example #3: Four Gaussians

Tuned Neural Network (hidden=2, activation=logistic)
Example #4: Two Pockets
Example #4: Two Pockets
Example #4: Two Pockets

K-NN (k=5, metric=euclidean)
Example #4: Two Pockets

Tuned Neural Network (hidden=2, activation=logistic)
Example #4: Two Pockets

Tuned Neural Network (hidden=3, activation=logistic)
Example #4: Two Pockets

Tuned Neural Network (hidden=4, activation=logistic)
Example #4: Two Pockets

Tuned Neural Network (hidden=10, activation=logistic)
BUILDING DEEPER NETWORKS
Q: How many layers should we use?
Deeper Networks

Q: How many layers should we use?
Q: How many layers should we use?

Deeper Networks
Q: How many layers should we use?

• Theoretical answer:
  – A neural network with 1 hidden layer is a universal function approximator
  – Cybenko (1989): For any continuous function $g(x)$, there exists a 1-hidden-layer neural net $h_\theta(x)$ s.t. $|h_\theta(x) - g(x)| < \epsilon$ for all $x$, assuming sigmoid activation functions

• Empirical answer:
  – Before 2006: “Deep networks (e.g. 3 or more hidden layers) are too hard to train”
  – After 2006: “Deep networks are easier to train than shallow networks (e.g. 2 or fewer layers) for many problems”

Big caveat: You need to know and use the right tricks.
Feature Learning

- **Traditional feature engineering:** build up levels of abstraction by hand
- **Deep networks** (e.g. convolution networks): learn the increasingly higher levels of abstraction from data
  - each layer is a learned feature representation
  - sophistication increases in higher layers

Figures from Lee et al. (ICML 2009)
Feature Learning

- **Traditional feature engineering**: build up levels of abstraction by hand
- **Deep networks** (e.g. convolution networks): learn the increasingly higher levels of abstraction from data
  - each layer is a learned feature representation
  - sophistication increases in higher layers

Figures from Lee et al. (ICML 2009)
Feature Learning

- **Traditional feature engineering:** build up levels of abstraction by hand
- **Deep networks** (e.g. convolution networks): learn the increasingly higher levels of abstraction from data
  - each layer is a learned feature representation
  - sophistication increases in higher layers

Figures from Lee et al. (ICML 2009)